

Watchdog processors in parallel systems

András Pataricza^{+,++}, István Majzik⁺⁺, Wolfgang Hohl⁺, Joachim Hönig⁺

⁺Institut für Mathematische Maschinen und Datenverarbeitung III, Universität Erlangen-Nürnberg, Martensstr. 3, D-91058 Erlangen, Germany

⁺⁺Dept. of Measurement and Instrument Engineering, Technical University of Budapest Műegyetem rkp. 9., H-1502 Budapest, Hungary

A watchdog processor (WDP) is a relatively simple coprocessor built for concurrent, information compaction based error detection in the main program control flow. A new algorithm called SEIS (Signature Encoded Instruction Stream) is presented for assigning signatures to high level-instructions. The main idea of this method is to embed the information necessary to the program flow check into the signatures themselves, thus avoiding large reference databases in the WDP and allowing high operational speed. Solutions for a fault-tolerant multiprocessing and multi-tasking implementation are described as well.

1. INTRODUCTION

A massively parallel multiprocessor contains several thousands of processing nodes. Yet, computing intensive applications still require extremely long execution times - weeks or even months. Moreover, the increasing number of processors can drastically reduce reliability. Thus, fault-tolerance becomes a key design factor. This requirement can be met by distributed memory MIMD (multiple-instruction multiple-data) systems.

1.1. The MEMSY Architecture

The new experimental multiprocessor MEMSY (Modular Expandable Multiprocessor System) developed at the University of Erlangen-Nuremberg serves both as a test-bed for high performance scientific computing and effective fault tolerance methods [1]. The system has a hierarchical, scalable, regular structure, with locally shared memory and a distributed operating system. At each level the processor nodes form a four-neighbor toroidal mesh. Nodes are coupled by multiport memories

through a fault tolerant interconnection network allowing fast data exchange.

In order to increase the computing power each node consists of four MC88k processors. Internally the processors have MMU chips containing local instruction and data caches as well. Memory is interfaced on a high speed dedicated bus. Peripherals are coupled via a VME bus.

The operating system of MEMSY is based on UNIX. Each node has its own local operating system kernel with basic services: e.g. administration of objects, scheduling, communication between objects and memory management. Fault tolerance services are based on a backward recovery scheme. This method uses periodical backups of all data necessary to restart from an intermediate point of the computation after a system crash or a detected non-fatal error. A vital requirement is the correctness of the saved data used for recovery, thus fast error detection is necessary. Moreover, a long error latency may prevent a proper fault diagnosis by the weak correlation between the fault and its functional error symptoms [2].

This research is part of the Hungarian-German Joint Scientific Research Project #70 with additional support from: SFB 182 (DFG), Konrad Zuse Program (DAAD), OTKA-760,T-3394 and F7414 (Hungarian NSF)

2. WATCHDOG PROCESSORS

The majority of computer failures results from transient faults. According to both previous experience and accelerated fault injection experiments about 50-60% of this faults are manifested as disturbances in the program control flow. Since the early eighties the most promising solution for checking the program execution flow in the main processor is the use of *watchdog processors* (WDP) [3]. A WDP, implemented as a relatively simple coprocessor, compares precomputed reference signatures with run-time signatures, which encode some characteristics of the reference program flow and the actual program flow, respectively.

2.1. Main approaches

Control flow checking can be classified depending on the run-time signature generation method used.

Originally, in most WDP methods the instruction fetch sequence on the system bus was checked (*derived signature* based methods) by using some kind of information compaction. However, in modern computer architectures the observability of the system bus is drastically reduced, e.g. by the use of on-chip caches and instruction prefetch queues.

Nowadays the so-called *assigned signature* based approach is almost exclusively used in computers based on off-the shelf components. In this method a preprocessor extracts the *program control-flow graph* (CFG) from the high level programming language source code. In the CFG vertices represent branch free program blocks (instruction sequences) and edges correspond to control transfers (e.g. branch-type instructions as IF_THEN_ELSE or CASE statements). An unambiguous signature is assigned to each vertex. Signature transfer statements are inserted into the source code.

During the main program run this signatures are transferred to the WDP uniquely identifying the program location. The WDP checks concurrently the correct execution of the main program by checking the received signature sequence. This sequence will be accepted as correct, if it corresponds to an existing

path in the program graph, independently of the semantic correctness of the branch selections. In the case of a conditional branch instruction, it is only checked whether the target instruction belongs to the set of the successors, but the selection itself remains un-checked.

2.2. Implementations

There are two traditional approaches for the implementation of signature assignment based watchdog processors:

In the first method published, the *signature integrity checking* (SIC) [4] a general-purpose microcomputer serves as WDP, for which the SIC-preprocessor extracts a *CFG checking program* in the same high level programming language as the main program e.g. by substituting branch-free program blocks with receive-signature instructions. The main drawbacks of this method are the high complexity and low speed of the WDP and its inability to handle function calls through pointers or interrupts.

As alternative for MEMSY initially a new, so-called *Extended Signature Integrity Checking* (ESIC) method was developed [5]. The program control-flow graph (CFG) is explicitly extracted from the source code by a preprocessor. Each subroutine is mapped to a separate subgraph. The generated signatures contain a field uniquely identifying the subroutine. Special signatures mark the start and end vertices of subroutines (SOP and EOP respectively). Before the start of the main program a *tabular representation of the CFG* (the set of the adjacency matrices of the subgraphs in a sparse matrix format) is downloaded into the WDP, defining for each signature (state) the set of the allowed successors.

In order to handle function calls the WDP is implemented as a finite deterministic stack automaton. By receiving an SOP signature the actual state is pushed onto stack and the WDP switches over to the table of the called subroutine. If it receives an EOP signature, it checks whether this signature belongs to the current subroutine and if so, the WDP resumes checking the calling subroutine by popping the saved state from the stack.

The experiences from previous experiments show an excellent fault coverage, but a necessity for the reduction of both the hardware complexity of the WDP (a transputer based microcomputer similar in complexity as the main processor itself) and of the time overhead related to signature processing as well.

3. THE SEIS METHOD

In the assigned signature method the only requirement for the labelling of instructions with signatures is their uniqueness for identifying the main program location. The main idea in the new method called *Signature Encoded Instruction Stream* (SEIS) is a different encoding algorithm of the CFG signatures, so that *each signature uniquely identifies its successors as well*, similarly to the fault-tolerant hardware implementation of finite state machines [6]. In this way only the last signature in each subgraph is to be stored in the WDP, as the check of the signature sequence requires only the combinational comparison of the actual signature and the successor fields in the previous signature. The program or automaton table handling in the previous methods can be omitted, reducing both hardware and time complexity of the WDP. Subroutine calls can be handled in an identical way as in ESIC.

3.1. The basic idea

A main difference to the finite state machine synthesis with an unlimited number of potential successors and predecessors of a state is, that in a CFG this number is very small for the overwhelming majority of instructions. Accordingly, when limiting the number of successors to a value k , a vertex can be identified by the concatenated labels of its successors. In the mathematical sense this is a sparse representation of the row in the adjacency matrix corresponding to the current vertex. In the case of multidirectional control transfer instructions violating this assumption, like a CASE statement, intermediate vertices are to be inserted into the CFG. The state transition has to be performed in multiple

steps. This requires only a moderate time overhead, as the number of the necessary additional states to a vertex depends only logarithmically on the number of edges. (When arranging the additional vertices as a k -ary tree of a depth d , the maximal number of the successors or predecessors can reach k^{d+1} .) In our implementation $k=3$ was chosen.

If we order the codes used for labelling and encode the vertices on a directed path in the CFG with subsequent codes (further referred to as sublabels), then the execution of the instruction sequence corresponding to this path will produce an easy-to-check sublabel stream consisting of subsequent codes.

A vertex can belong to multiple paths, and accordingly, multiple (maximally k) sublabels can be assigned to it. As signature we will use their concatenation. If a vertex is traversed by less than k paths, then a dummy sublabel (not belonging to any vertex) is assigned to the remaining part of the signature in order to have a fixed-length encoding. During program execution the WDP has only to check whether some sublabel in the current signature is a successor of a sublabel in the previous one.

3.2. The encoding algorithm

The last problem to be solved is the extraction of the directed edge sequences from the CFG, which can be reduced to the well-known Eulerian circuit problem, i.e. to find a circuit in a graph, which contains each edge exactly once. Such a circuit exists if and only if each vertex in the graph has same numbers of both incoming and outgoing edges.

In the algorithm at first the CFG is completed with additional edges to an Eulerian graph. After generating the Eulerian circuit it is partitioned into separate edge trails by removing the additional edges. Successive sublabels are assigned to the vertices in a trail. The second successor of the last sublabel in the previous trail is attached to the starting element of the next trail, in this way the sublabel sequences remain disconnected by an unused sublabel. Finally, signatures are generated as the concatenation of the subroutine (function) code and the sublabels, eventually extended

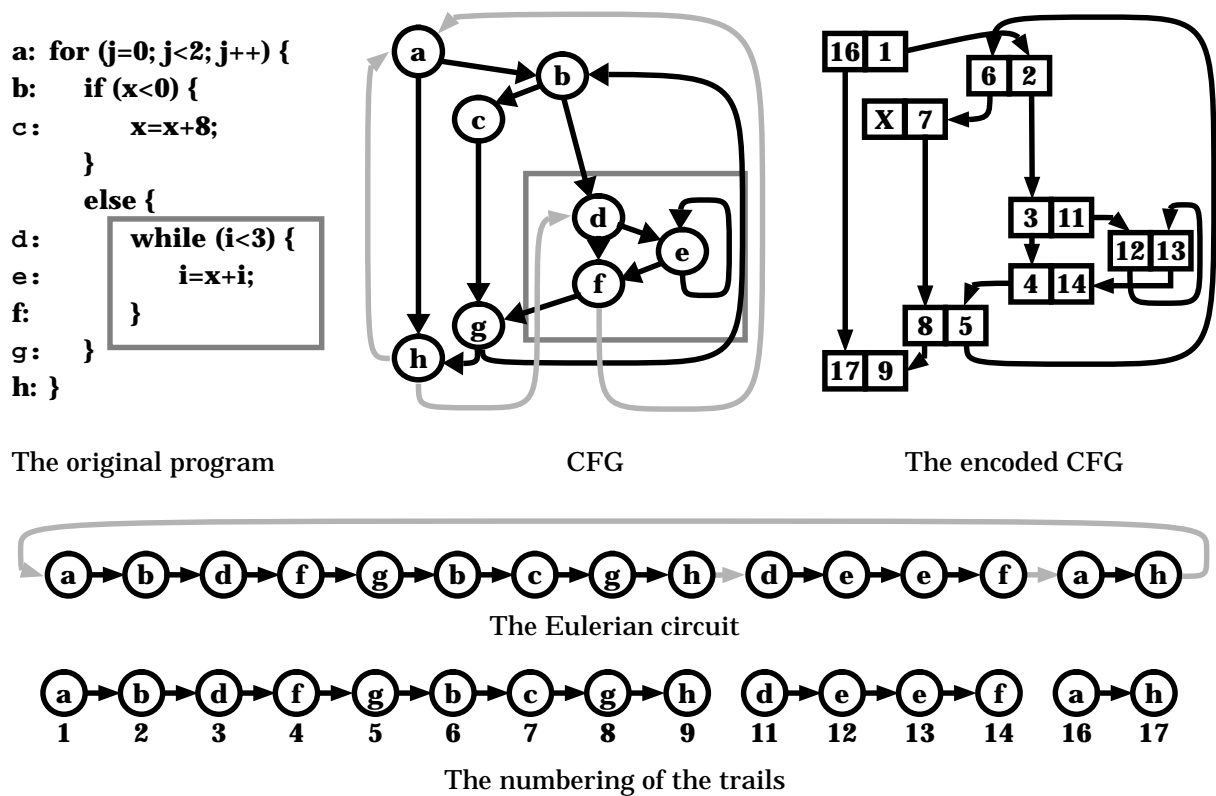


Fig. 1. The encoding algorithm

with dummy sublabels to k fields.

An example is shown in Fig.1. From the C program source the corresponding CFG is generated. The addition of the edges $h \rightarrow d$, $f \rightarrow a$ and $h \rightarrow a$ (marked by a grey color) transforms it to an Eulerian graph. After finding the Eulerian circle shown below, it is cut to trails by removing the additional edges. If the sublabels are ordered as positive integers, the sublabels 1..9, 11..14, 16..17 are assigned to the vertices in the trails. The resulting encoded CFG is shown in the right upper corner. For simplicity $k=2$ was assumed. The X in the signature attached to vertex b denotes a dummy sublabel field. In our case this field may contain 7 once again or for example 19, a sublabel neither the predecessor nor the successor of any other sublabel assigned to a vertex.

If this program segment is entered with $x=j=1$, then the program flow control flow will be the following one: $\langle a, b, d, e, e, f, g, b, d, f, g, h \rangle$.

The corresponding signature sequence is:

$[16,1], [6,2], [3,11], [12,13], [12,13], [4,14], [8,5], [6,2], [3,11], [4,14], [8,5], [17,9]$. In this sequence those sublabels are marked with a bold typesetting, which are a valid successor of the previous signature (the initial value in the WDP is set to 0 at the entry point of a program or function).

It should be pointed out, that the elementary sublabel assignment construction depends only of the instruction to be labelled (as denoted in Fig. 1. for the WHILE instruction) and the algorithm described above can be merged with the syntax analysis in the SEIS-preprocessor by using an elementary graph labelling building block library.

4. IMPLEMENTATION OVERVIEW

In a WDP the length of the signature code words is practically limited by the bus width. In order to keep the time overhead related to

signature transfers on an acceptable level, multiple instructions or bus cycles per signature transfer are to be avoided. To achieve a sufficient bandwidth, information is transferred through the address bus as well. The prototype of the WDP is memory mapped through the communication memory interface.

4.1. Support of multitasking

In a multitasking environment the WDP must be shared by emulating for each task an independent signature checker. The simplest solution for this is the concatenation of a task (process) ID field to the signatures. In the WDP for each process a private stack has to be allocated. The signature sequences can be separated according to their task ID field.

Task IDs are generated by the operating system only at the creation phase of a new task. A static task ID insertion would mean a full reediting of the binary code. Moreover, this solution would not allow code sharing between different processes, as they would differ in the ID fields of the signatures.

Accordingly, a dynamic insertion of the task IDs was chosen. As task IDs are encoded into the address information of the signature transfer bus cycle, it undergoes a virtual to physical address transformation in the MMU, as shown in Fig.2. During task initialization the MMU is programmed in such a way, that the physical address bits correspond exactly to

the task ID. In doing so, different tasks can transfer signatures to the WDP through different physical address ranges while having identical virtual address bits in the task ID field of the signature in the binary code. In case of code sharing the same virtual WDP addresses can be referenced. The replication of code segments is avoided.

Another major problem results from the sharing of the WDP stack as a common resource between different processes. A static stack partitioning strategy would require a storage of unacceptable size, corresponding to the product of the number of tasks and the maximal allowed stack size per task. As not all tasks use the whole allowed stack area, the stack is partitioned dynamically using a linked list organization.

4.2. Multiprocessor support

In SEIS the signature processing time is as low as a single bus cycle. Signatures are sent only at the beginning of a high level language instruction, the excess speed allows sharing the WDP between different nodes of a multiprocessor. As interprocessor communication in MEMSY is based on memory sharing, the WDP is integrated into this communication system, serving five computing nodes, each consisting of 4 processors. The actual node and processor IDs are derived from the bus grant signals of the node system bus and the ac-

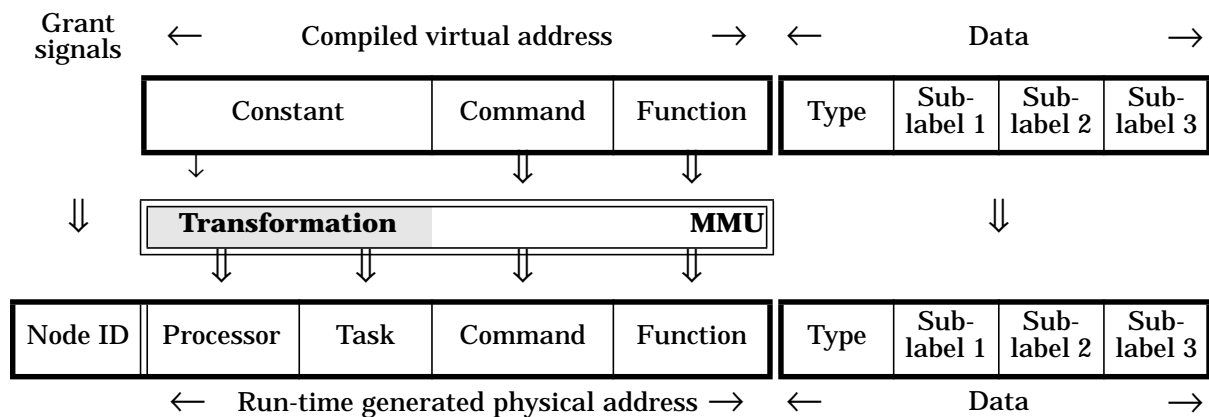


Fig.2. The transformation of the signatures

knowledge signals of the arbiter of the multi-port communication memory. This IDs are concatenated by the WDP hardware to the signature.

4.3. Support of error recovery

Another important feature of the WDP is the support of error recovery. As described earlier, in MEMSY a backward recovery strategy is used. During the computations intermediate checkpoints are saved. In the case of an error the system is restarted from this stored state avoiding the loss of the whole computing time. In order to always have a valid checkpoint, a two phase commit protocol is used in MEMSY.

The WDP supports the checkpoint-based error recovery in two ways:

- Simultaneously to saving checkpoint data in the main processor, the corresponding WDP state is saved internally, too. In the case of program recovery this state is restored autonomously by the WDP.
- The "save WDP checkpoint state" command is embedded into the signature sequence. Moreover, the address range of the WDP corresponding to this command can only be accessed in privileged mode to avoid incidental checkpoint overwrite due to an error. Accordingly, it is possible to allow checkpoint generation only at predefined main program locations.

5. EXPERIMENTAL RESULTS

An experimental version of the SEIS WDP was built around AMD Mach 210 series FPGA circuits serving 5 nodes in MEMSY.

Signature checks in the current implementation of the SEIS method can be performed even during the signature transfer itself, only the checkpoint generation results in a moderate slow-down of the main program. Experiments have shown a typical static code length overhead of 5-10%, and about 15-25% of execution time, depending on the program under consideration.

First simulated fault injection based experiments in MEMSY show that the primary,

standard error detection mechanisms (segmentation violation, word alignment and illegal instruction checks) detect approximately 60-75% of all errors. Using the WDP increases the overall error coverage to 75-90%.

REFERENCES

1. Dal Cin, M. et al.: *Fault Tolerance in Distributed Shared Memory Multiprocessors*. To appear in Springer LNCS, 1993
2. Dal Cin, M. et al.: *Error Detection Mechanisms for Massively Parallel Multiprocessors*. Proc. Euromicro Workshop on Parallel and Distributed Processing, (1993), 401-408.
3. Mahmood, A; McCluskey, E.J.: *Concurrent Error Detection Using Watchdog Processors - A Survey*. IEEE TC, 37. 160-174, (1988)
4. Lu, D. J.: *Watchdog Processors and Structural Integrity Checking*. IEEE TC, 31. 681-685, (1982)
5. Michel, E.; Hohl, W.: *Concurrent Error Detection Using Watchdog Processors in the Multiprocessor System MEMSY*. Proc. 5th Intl. Conf. Fault-Tolerant Computing Systems, Informatik Fachberichte 283, 54-64, Springer, (1991)
6. Robinson, S.H.: *Finite-state Machine Synthesis for Continuous, Concurrent Error Detection Using Signature-invariant Monitoring*. Research Report CMUCAD-92-36, Carnegie Mellon University (1992)